Submit (on Gradescope) a 3-page, single-spaced report in PDF format describing details regarding the steps you followed for developing the classifier for predicting the product review sentiments.  
Be sure to include the following in the report:  
1. Your identifier as registered on the miner website.  
2. Rank & accuracy score for your submission (at the time of writing the report).  
3. A detailed and clear description of your approach, and how and why you chose the parameters, features, distance measure, etc.  
4. Any graphs or tables illustrating key experiments you did in the process of choosing your final model.

From the beginning, I had planned to use a bag of words approach to organizing the provided training data by collecting all the relevant words in a list. I planned on merging all vectors if the sentiment of the reviews matched (positive reviews merged with other positive reviews, negative reviews merged with other negative reviews) and doing a frequency count instead of a binary appearance count. In technical terms, I created two python dictionaries / hash maps (one for positive sentiment and one for negative) and mapped a frequency count (or degree as written in my code) to each word found in the training file based on the sentiment given in the training file. This would allow me to create a variant of the K-nearest model that would consider all the neighbors of the dimension/word but puts a degree of trust in the neighbors that are closest (with the higher frequency and weight)

At first attempt, my accuracy score was much higher than I anticipated (at a 0.80). I was surprised because I submitted the results of a barebones approach to the assignment. In the first draft code, I considered all words (capitalization matters). I used a simple weight scheme to determine if a word from a review would be considered in the classification algorithm. The code would read a word from the review and it would find the percentage of positive over the negative counts from the training phase (inverted if the denominator were higher) so that the percentage would equate to a real number less than 1 and greater than -1, and the code would then sum up all of the word ratios in the review to see if the sum is a positive or negative result. This sort of variation of the K-nearest implementation takes in all words that occurred in the training and plotted the current test review data to do a K-nearest analysis against all recorded training points (with the appropriate weights). This means that the algorithm during the test review would consider all the words within the test review if the word were an element of the intersection between the current test review “document” and the total set of training data (represented by two dictionaries / hash maps).

I decided to expand my code by implementing feature subset selection. The simplest way I thought was to suppress any redundant words during training, and I found a nice list of stop words here: <https://countwordsfree.com/stopwords>. With this list, I was able to utilize feature subset selection by removing noise from the training subset. However, with each new attempt (adding more and more stop words), I found myself getting lower scores each time. This made me call back to the lecture when the professor explained that bag of words is strong because of its simplicity. I ended up reducing the list of stop words to only two words: “THE” and “A”. Here, I realized I made a simple mistake of including redundant words with different capitalizations. I quickly changed my code to ignore capitalization. This ended up stopping my score from dropping even further past a 0.49 (the last feature subset selection code attempt resulted in a 0.54). Because increasing the complexity of the preprocessing of inputs seemed to be worse for the current K-nearest algorithm, I decided that I had to create a better K-nearest algorithm approach instead of trying to decrease the dimensionality in order to improve my accuracy rating.

I made the algorithm for K-nearest more consistent by making it simpler. Instead of a percentage-based weight, I changed it to a ratio of the higher count over the lower count. An example of this would be if a word in the document were “terrible”, then the amount of times that word would appear in the negative training set would be a big number such as 100 negative occurrences, and the amount of times that word would appear in the positive training set would be a small number such as 2 positive occurrences. The ratio that the algorithm would produce would be 50/1 which would then add 50 to the review sentiment summation. This method of weighting is a sort of soft weighing system where the training rating as well as the current testing review frequency of a word has an effect on the weight (though the length of the current testing review document would not matter). I also added a variation of the document frequency threshold (which would be an example of feature subset selection) where if the current word’s ratio fell short of the threshold, then the algorithm would not consider it in the total sentiment rating. At this point, I managed to reach 0.82 accuracy (though there were some hiccups after some threshold testing).